SPARC
สถาปัตยกรรมที่มาแรง

ความเป็นมา
และปรัชญาของสถาปัตยกรรม
แบบ RISC และ SPARC

บทความนี้จะเป็นการกล่าวถึงสถาปัตยกรรมของ RISC (Reduce Instruction Set Computer) กับ SPARC (Scalable Processor ARChitecture) อย่างคร่าว ๆ โดยในช่วงแรกจะเป็นการกล่าวถึงประวัติความเป็นมาและปรัชญาของสถาปัตยกรรมแบบ RISC ช่วงที่สองเป็นการกล่าวถึงสถาปัตยกรรมและชุดคำสั่งของ SPARC และในช่วงสุดท้ายเป็นการกล่าวถึงเทคนิคบางประการที่นำมาใช้ในการเพิ่มสมรรถนะของโปรเซสเซอร์ เช่น การเพิ่มฟังก์ชันยูนิต (function unit) เพื่อทำให้โปรเซสเซอร์สามารถเอ็กซีคิวต์ได้มากกว่า 1 คำสั่งใน 1 คาบเวลา (clock cycle) เป็นต้น

RISC

ต้นกำเนิดของสถาปัตยกรรมแบบ RISC ก็คือเครื่อง CDC 6600 ของ Control Data Corporation ซึ่งผู้ออกแบบคือ Thunton, Cray และผู้ร่วมงานของพวกเขาได้ตระหนักถึงการทำชุดคำสั่งให้ง่ายในการสร้างเครื่องคอมพิวเตอร์ความเร็วสูง หลังจากนั้น Cray ก็ได้ออกมาตั้งบริษัทของตัวเองที่มีชื่อว่า Cray Research และยังคงแนวความคิดในลักษณะเดียวกันคือ "คงไว้ซึ่งความง่ายของเครื่องคอมพิวเตอร์" ในการสร้าง CRAY-1 แต่ความเป็น RISC ของเครื่อง CDC 6600 ก็มิได้มีผลกระทบโดยตรงกับสถาปัตยกรรมแบบ RISC ในปัจจุบัน โครงการ

วิจัยที่มีผลกระทบต่อสถาปัตยกรรมแบบ RISC มี 3 โครงการใหญ่ ๆ คือ โครงการ 801 ของ IBM, โครงการโปรเซสเซอร์ RISC ที่ University of California, Berkeley และโครงการ MIPS (Microprocessor without Interlocked Pipeline Stages) ที่ Stanford University

โครงการ 801 ได้เริ่มขึ้นในปี ค.ศ. 1975 และเสร็จสิ้นในปี ค.ศ. 1979 งานวิจัยนี้เกิดขึ้นที่ IBM Thomas J. Watson Research Laboratory เนื่องจากผู้เขียนไม่มีเอกสารที่เป็นรายละเอียดเกี่ยวกับโครงการ 801 จึงไม่สามารถขยายความได้มากนัก ทราบแต่เพียงว่าเป็นโครงการที่สร้างเครื่องมินิคอมพิวเตอร์ โดยใช้เทคโนโลยี ECL (Emitter Couple Logic) มีการทำงานโดยใช้ชุดคำสั่งแบบง่าย ๆ มีรีจิสเตอร์ขนาด 32 บิตจำนวน 32 รีจิสเตอร์ และใช้เทคนิคของตัวแปลภาษา (compiler) เข้าช่วยเพื่อให้สามารถประมวลผลได้เร็วขึ้น

โครงการโปรเซสเซอร์ RISC กำเนิดมาจากแนวความคิดของ Patterson และ Sequen ในช่วงฤดูหนาวปี ค.ศ. 1980 หลังจากนั้นก็ได้เริ่มศึกษาและออกแบบจนเสร็จในช่วงฤดูใบไม้ผลิในปี ค.ศ. 1981 การออกแบบจะยึดหลักสำคัญ 4 ประการคือ

ก. พยายามทำให้การทำงานของแต่ละคำสั่งใช้ 1 คาบเวลา

ข. ทำให้ทุกคำสั่งมีขนาดและรูปแบบที่แน่นอน

ค. การติดต่อกัน หน่วยความจำใช้คำสั่ง LOAD และ STORE เท่านั้น (ลักษณะเช่นนี้เรียกว่า LOAD/STORE MACHINE) และการกระทำที่เหลือ เช่น การบวก, การกระทำเชิงอรรถ จะกระทำระหว่างรีจิสเตอร์

ง. การออกแบบต้องการให้สนับสนุนการทำงานของภาษาชั้นสูง สำหรับ RISC ตัวนี้เลือกใช้ภาษา C

INSTRUCTION	OPERANDS	COMMENTS	
ADD	Rs.S2.Rd	Rd – Rs + S2	integer add
ADDC	Rs.S2.Rd	Rd – Rs + S2 + carry	and with carry
SUB	Rs.S2.Rd	Rd – Rs – S2	integer subtract
SUBC	Rs.S2.Rd	Rd – Rs – S2 – carry	subtract with carry
SUBR	Rs.S2.Rd	Rd – S2 – Rs	integer subtract
SUBCR	Rs.S2.Rd	Rd – S2 – Rs – carry	subtract with carry
AND	Rs.S2.Rd	Rd – Rs & S2	logical AND
OR	Rs.S2.Rd	Rd – Rs | S2	logical OR
XOR	Rs.S2.Rd	Rd – Rs xor S2	logical EXCLUSIVE OR
SLL	Rs.S2.Rd	Rd – Rs shifted by S2	shift left
SRL	Rs.S2.Rd	Rd – Rs shifted by S2	shift right logical
SRA	Rs.S2.Rd	Rd – Rs shifted by S2	shift right arithmetic
LDL	(Rx)S2.Rd	Rd – M[Rx + S2]	load long
LDSU	(Rx)S2.Rd	Rd – M[Rx + S2]	load short unsigned
LDSS	(Rx)S2.Rd	Rd – M[Rx + S2]	load short signed
LDBU	(Rx)S2.Rd	Rd – m[Rx + S2]	load byte unsigned
LDBS	(Rx)S2.Rd	Rd – M[Rx + S2]	load byte signed
STL	Rm.(Rx)S2	M[Rx + S2] – Rm	store long
STS	Rm.(Rx)S2	M[Rx + S2] – Rm	store short
STB	Rm.(Rx)S2	M[Rx + S2] – Rm	store byte
JMP	COND.S2(Rx)	pc – Rx + S2	conditional jump
JMPR	COND.Y	pc – pc + Y	conditional relative
CALL	Rd.S2(Rx)	Rd – pc. next	call
		pc – Rx + S2. CWP – CWP – 1	and change window
CALLR	Rd.Y	Rd – pc. next	call relative
		pc – pc + Y. CWP – CWP – 1	and change window
RET	Rm.S2	pc – Rm + S2. CWP – CWP + 1	return and change window
CALLINT	Rd	Rd – last pc: next CWP – CWP – 1	disable interrupts
RETINT	Rm.S2	pc – Rm + S2: next CWP – CWP + 1	enable interrupts
LDHI	Rc.Y	Rd<31:13> – Y: Rd<12.0> – 0	load immediate high
GTLPC	Rd	Rd – last pc	to restart delayed jump
GETPSW	Rd	Rd – PSW	load status word
PUTPSW	Rm	PSW-Rm	set status word

รูปที่ 1 ชุดคำสั่งของ RISC I

RISC I ถูกสร้างเสร็จในปี ค.ศ. 1982 เป็นโปรเซสเซอร์ขนาด 32 บิต สถาปัตยกรรมภายในประกอบด้วย ALU (Arithmetic Logic Unit), PC (Program Counter), PSW (Program Status Word) และที่สำคัญก็คือ รีจิสเตอร์ขนาด 32 บิต จำนวน 78 รีจิสเตอร์ มีคำสั่งทั้งหมด 31 คำสั่ง ดังแสดงในรูปที่ 1 ลักษณะของคำสั่งแบ่งออกเป็น 4 แบบคือ คำสั่งทางด้านคณิตศาสตร์และตรรกศาสตร์ คำสั่งการเข้าถึงหน่วยความจำ, คำสั่งควบคุม เช่น คำสั่ง branch และคำสั่งทั่ว ๆ ไป การอ้างแอดเดรสมีแบบเดียวคือ ใช้ index บวกกับ displacement แต่การอ้างแบบนี้สามารถสังเคราะห์การอ้างแบบอื่น ๆ ได้ดังแสดงในรูปที่ 2 และในปี ค.ศ. 1983 ก็ได้สร้าง RISC II ซึ่งเป็น LOAD/STORE MACHINE

ADDRESSING	VAX	RISCEQUIVALENT
REGISTER	Rx	Rx
IMMEDIATE	#LITERAL	S2 (13-BIT LITERAL)
INDEXED	Rx + DISPL	Rx + S2 (13-BIT DISPLACEMENT)
ABSOLUTE	@#ADDRESS	r0 + S2 (r0 ≡ 0)
REG INDIRECT	(Rx)	Rx + 0

รูปที่ 2 โหมดในการอ้างแอดเดรสของ RISC I เมื่อเทียบกับ VAX

เช่นเดียวกับ RISC I แต่มีคำสั่ง 39 คำสั่งและใช้พื้นที่ของชิลิกอนลดลง 25% จากของเดิม RISC I และ RISC II ใช้เทคนิคของสถาปัตยกรรมทางฮาร์ดแวร์ที่เรียกว่า register window ร่วมกับตัวแปลภาษาแบบธรรมดา เพื่อช่วยในการเพิ่มสมรรถนะของโปรเซสเซอร์

โครงการ MIPS เป็นโครงการที่อยู่ภายใต้การดำเนินงานของ Hennessy และผู้ร่วมงาน โดยมีแนวความคิดในการเพิ่มสมรรถนะของโปรเซสเซอร์แตกต่างไปจากโครงการ RISC I และ II คือพยายามทำให้ pipeline ทำงานมีประสิทธิภาพมากขึ้น โดยใช้ตัวแปลภาษาช่วยในการจัดเรียงคำสั่ง ที่เป็นเช่นนี้ก็เพราะว่าจากการศึกษาพบว่า ปัญหาที่มีผลกระทบต่อการทำงานของ pipeline มี 3 ประการ คือ

ก. ปัญหาอันเนื่องมาจากการพยายามใช้ทรัพยากรร่วมกัน เมื่อฮาร์ดแวร์ไม่สามารถช่วยให้การทำงานแบบ overlapped เกิดขึ้นพร้อม ๆ กันได้

ข. ปัญหาอันเนื่องมาจากการขึ้นต่อกันของข้อมูล เมื่อ operand ของคำสั่งที่ตามมาขึ้นอยู่กับผลลัพธ์ที่ได้จากคำสั่งก่อนหน้านี้ ทำให้คำสั่งที่ตามมาไม่สามารถประมวลผลได้

ค. ปัญหาอันเนื่องมาจากคำสั่งที่เปลี่ยนแปลงสถานะ (state) ของโปรเซสเซอร์ เช่น คำสั่ง branch เป็นต้น ทำให้การทำงานของ pipeline เกิดการสูญเปล่า

จากปัญหาดังกล่าวจำเป็นที่ต้องใช้ตัวแปลภาษาที่มีความสามารถมาก ซึ่งเรียกโดยทั่ว ๆ ไปว่า optimizing compiler โปรเซสเซอร์ MIPS เป็น LOAD/STORE MACHINE เช่นเดียวกับ RISC I และ II แต่สิ่งที่แตกต่างกันอย่างชัดเจนคือ รีจิสเตอร์ของ MIPS มีเพียง 16 รีจิสเตอร์เท่านั้น ทั้งนี้ก็เพราะแนวความคิดในการแก้ปัญหาที่แตกต่างกันนั่นเอง

โปรเซสเซอร์ RISC มีความแตกต่างจากโปรเซสเซอร์ CISC (Complex Instruction Set Computer) ซึ่งสามารถแบ่งได้เป็น 8 ข้อดังนี้

ก. RISC มีคำสั่งที่ง่ายสามารถกระทำให้เสร็จได้ภายใน 1 คาบเวลา แต่ในแง่ของ CISC แล้วจะใช้หลายคาบเวลา และคำสั่งมีความสลับซับซ้อนมาก เช่นคำสั่งในการอ้างหน่วยความจำเป็นต้น

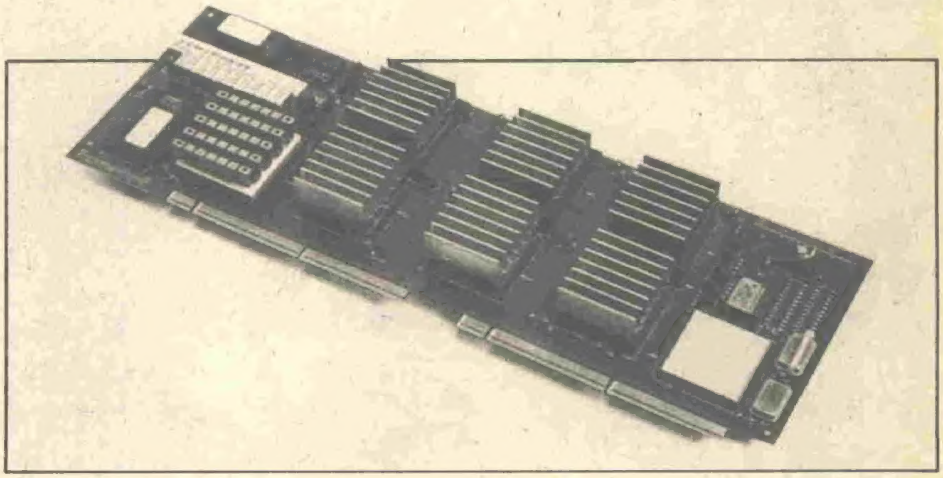
ข. RISC มีเฉพาะคำสั่ง LOAD และ STORE เท่านั้นที่สามารถอ้างถึงหน่วยความจำได้โดยตรง สำหรับ CISC แล้วมีหลายคำสั่งที่สามารถอ้างถึงตำแหน่งของหน่วยความจำได้โดยตรง เช่น การบวกค่าในรีจิสเตอร์กับค่าในหน่วยความจำ เป็นต้น

ค. RISC เน้นการทำงานโดยใช้เทคนิค pipeline เพราะเป็นเทคนิคที่ช่วยให้มีอัตราปริมาณงานเพิ่มขึ้น สำหรับ CISC แล้วจะไม่เน้นที่ประเด็นนี้มากนัก แต่ในปัจจุบันโปรเซสเซอร์แบบ CISC ก็ใช้เทคนิค pipeline เช่นกัน แต่การแบ่ง stage ของ pipeline ค่อนข้างจะน้อยกว่า

ง. RISC มีส่วนที่ควบคุมการทำงานภายในโปรเซสเซอร์เป็นแบบ hardwire ที่เป็นเช่นนี้ได้ก็เพราะว่า RISC มีชุดคำสั่งที่ไม่สลับซับซ้อน ทำให้การออกแบบ hardwire ไม่มีความซับซ้อนตามไปด้วย และข้อดีของ hardwire ก็คือ มีความเร็วสูงกว่าส่วนควบคุมที่เป็นแบบ microprogram ซึ่ง CISC จำเป็นต้องใช้เพราะมีชุดคำสั่งที่สลับซับซ้อน ทำให้ต้องการส่วนควบคุมที่มีความสลับซับซ้อนตามไปด้วย

จ. RISC มีชุดคำสั่งที่มีรูปแบบและขนาดที่แน่นอน ทำให้ง่ายต่อการถอดรหัส เพราะการ fetch คำสั่งเข้ามา 1 ครั้งสามารถถอดรหัสได้ทันที แต่ในแง่ของ CISC แล้วจะมีชุดคำสั่งที่มีรูปแบบที่แตกต่างกันและมีขนาดไม่แน่นอน ซึ่งทำให้การ fetch 1 ครั้ง ไม่ได้คำสั่งที่สมบูรณ์

ฉ. RISC มีจำนวนคำสั่งและรูปแบบในการอ้างตำแหน่งของหน่วยความจำไม่มากนัก แต่มีเฉพาะที่สำคัญ ๆ และ



สามารถนำมาสังเคราะห์ให้เป็นรูปแบบต่าง ๆ ได้ชัด

ซ. RISC มีชุดของรีจิสเตอร์ค่อนข้างมาก ทั้งนี้เนื่องจากการศึกษาพฤติกรรมของโปรแกรมที่เขียนโดยใช้ภาษาระดับสูง พบว่ามีการเรียกโปรแกรมย่อยค่อนข้างมาก แทนที่จะเก็บสถานะต่าง ๆ ลงบนหน่วยความจำก็จะเก็บลงบนรีจิสเตอร์แทน เพราะการเข้าถึงรีจิสเตอร์สามารถกระทำได้รวดเร็วกว่า และไม่ไปรบกวนบัสของหน่วยความจำ และที่ RISC สามารถกระทำได้เช่นนี้ก็เพราะว่า ส่วนควบคุมถูกเปลี่ยนไปเป็น hardwire ทำให้มีเนื้อที่บนแผ่นซิลิกอนเหลือมากพอที่จะบรรจุรีจิสเตอร์ลงไปได้

SPARC คืออะไร

ในปี ค.ศ.1987 บริษัท ซันไมโครซิสเต็มส์ ได้เปิดตัวเวอร์กสเตชันรุ่นใหม่มีชื่อว่า SUN-4 หรือ SPARC station ไปพร้อม ๆ กับสถาปัตยกรรมแบบ SPARC โดยพยายามมุ่งหวังว่าจะทำให้เป็นสถาปัตยกรรมแบบเปิดซึ่งไม่ขึ้นอยู่กับผู้ผลิตชิพ ดังนั้นสถาปัตยกรรม SPARC จึงเป็นเพียงการกำหนดหน่วย (unit) หลัก ๆ และชุดคำสั่งเพื่อให้มีความเหมือนกันทางด้านคำสั่งที่ใช้ในการประมวลผล ซึ่งรวมถึงรหัสเครื่อง (machine code) ด้วย แต่ในแง่ของเทคโนโลยีหรือรายละเอียดทางด้านฮาร์ดแวร์ไม่มีการกำหนดเอาไว้ ดังนั้นจึงขึ้นอยู่กับผู้ผลิตชิพว่าจะใช้เทคโนโลยีอะไรและจะออกแบบฮาร์ดแวร์อย่างไร ในปัจจุบันได้มีหลายบริษัทที่ผลิตชิพ SPARC เช่น Cypress Semiconductor Bipolar Integrated Technology (BIT), LST logic, Texas Instrument (TS), Fujitsu, Solboume /Matsushita, Metaflow Technology เป็นต้น ซึ่งแต่ละบริษัทก็มีวิธีการที่แตกต่างกัน เช่น TI และ Metaflow พัฒนา Superscalar SPARC ขณะที่ BIT ใช้เทคโนโลยี ECL ในการผลิต SPARC (B5000) เป็นต้น สถาปัตยกรรม SPARC ที่จะกล่าวถึงนี้มีความคล้ายคลึงกับโครงการ RISC ซึ่งมีจุดเด่นคือ overlapped register

2.1 ลักษณะทั่วไปของ SPARC

สถาปัตยกรรม SPARC ประกอบด้วยหน่วยหลัก ๆ 4 หน่วยคือ CPU (Central Processing Unit) หรือเรียกโดยทั่วไปว่า IU (Integer Unit), FPU (Floating Point Unit)

SPARC เป็นสถาปัตยกรรมแบบ 32 บิตเต็ม และกำหนดให้ 1 เวิร์ดมีขนาด 32 บิต (8 บิตคือ 1 ไบต์ ซึ่งก็คือข้อมูลหรือคำสั่ง) ตามขอบเขตของเวิร์ด (word boundary) นั่นก็คือ การอ้างแอดเดรสจะเป็นจำนวนเท่าของ 4 การปรับเช่นนี้จะเป็นการเพิ่มประสิทธิภาพทางด้าน memory bandwidth การจัดเก็บวัตถุลงหน่วยความจำเป็นไปในลักษณะของบิตที่มีนัยสำคัญสูงสุด อยู่ในตำแหน่งที่มีนัยสำคัญน้อยที่สุดหรือเรียกโดยทั่วไปว่า big endian รีจิสเตอร์มีขนาด 32 บิต ทั้ง integer และ floating poing แต่คำสั่ง LOAD และ STORE สามารถกระทำกับข้อมูลขนาด 8, 16, 32 และ 64 บิตได้ กรณีที่เป็น 64 บิตจะใช้เป็นคู่รีจิสเตอร์ที่อยู่ติดกัน และคำสั่งที่ใช้มีขนาด 32 บิต

2.2 ลักษณะรีจิสเตอร์ของ SPARC

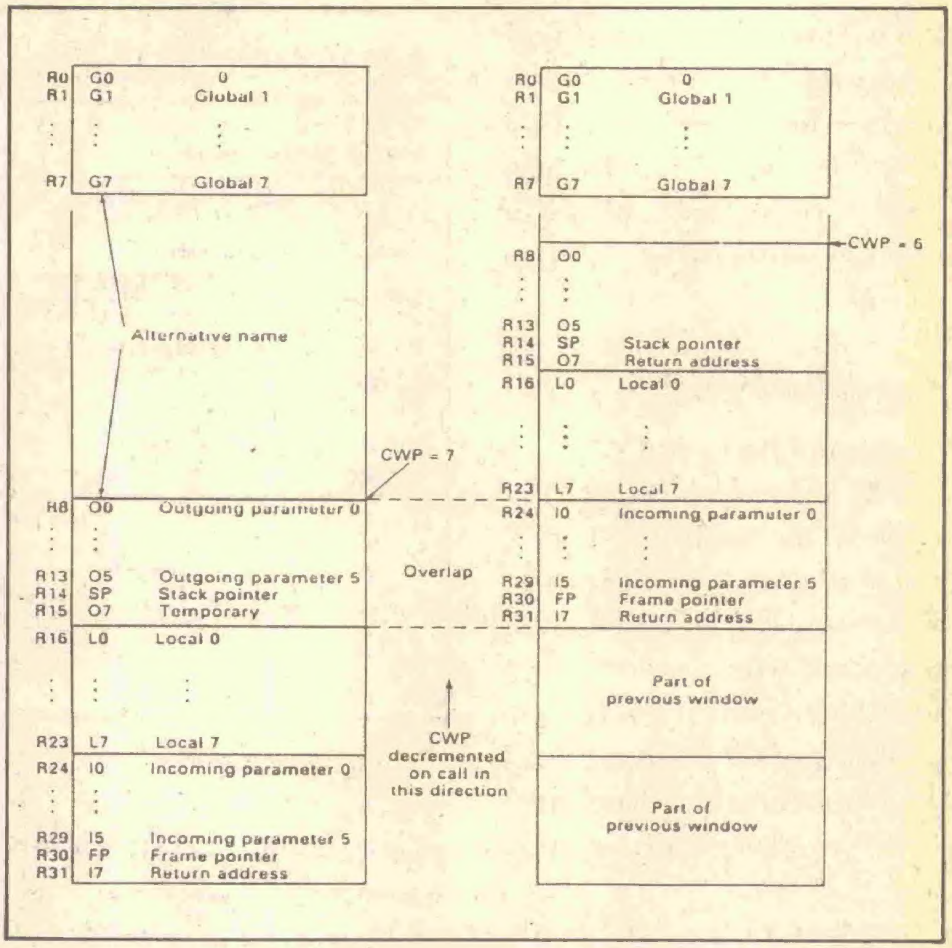
SPARC มีการจัดรูปแบบของรีจิสเตอร์ในลักษณะของวินโดว์ ที่มีการทับซ้อนกันบางส่วน (overlapped window) ดังแสดงในรูปที่ 3 แต่ที่ขณะใดขณะหนึ่งจะเห็นเพียงวินโดว์เดียวเท่านั้น ซึ่งมีจำนวนรีจิสเตอร์คงที่คือ 32 รีจิสเตอร์ โดยมี CWP (Current Window Pointer) เป็นตัวชี้วินโดว์ปัจจุบัน จำนวนรีจิสเตอร์ที่จะมีได้ไม่ถูก

กำหนดไว้ในสถาปัตยกรรม SPARC แต่มีการกำหนดให้ CWP มีขนาด 5 บิต ดังนั้นจำนวนรีจิสเตอร์จึงได้มีทั้งหมด 520 รีจิสเตอร์ (32×18+8) ภายใน 1 วินโดว์มีการแบ่งรีจิสเตอร์ออกเป็น 4 กลุ่ม คือ globals, outs, locals และ ins แต่ละกลุ่มมีจำนวนรีจิสเตอร์เท่า ๆ กัน ซึ่งอธิบายได้ดังนี้

globals เป็นกลุ่มรีจิสเตอร์ที่ใช้ร่วมกันในทุก ๆ วินโดว์ G0 ถูกกำหนดให้เป็นค่าคงที่เท่ากับ "0" G1-G7 ใช้เก็บค่าต่าง ๆ เช่น ตัวชี้ (pointer), เลขจำนวนเต็ม เป็นต้น

outs เป็นกลุ่มรีจิสเตอร์ที่ใช้ในการผ่านค่าระหว่าง โพรซีเดอร์ หรือ function call เช่น โพรซีเดอร์ A เรียกใช้โพรซีเดอร์ B โพรซีเดอร์ A จะนำค่าที่จะผ่านให้กับโพรซีเดอร์ B มาเก็บไว้ในกลุ่มรีจิสเตอร์ outs เมื่อ CWP ถูกเปลี่ยนค่าให้ไปชี้ที่ตำแหน่งของโพรซีเดอร์ B โพรซีเดอร์ B ก็จะมองเห็นกลุ่มรีจิสเตอร์ outs ของ A เป็นกลุ่มรีจิสเตอร์ ins (ซึ่งจะกล่าวถึงต่อไป) ของ B และนี่ก็คือการทับซ้อนระหว่างวินโดว์นี่เอง ค่าที่ใช้ในการส่งผ่านจะถูกเก็บในรีจิสเตอร์ 01-05 ส่วน 06 จะเก็บค่า SP (Stack Pointer) เพื่อชี้ไปยังหน่วยความจำสแต็ก ซึ่งถูกใช้ในกรณีที่คำสั่งที่ใช้ในการส่งผ่านมีเป็นจำนวนมาก เร็วกว่าจำนวนของรีจิสเตอร์ที่จะรับได้ 07 ใช้ในการเก็บค่าของตำแหน่งที่จะกลับมาอันเนื่องมาจากคำสั่ง CALL

locals เป็นกลุ่มรีจิสเตอร์ที่จะถูกใช้อย่างไรก็ได้ ตามแต่ผู้เขียนโปรแกรมหรือตัวแปรภาษาจะเห็นว่าเหมาะสม เช่น ใช้เก็บค่าตัวแปรต่าง ๆ และจะถูกใช้ได้เฉพาะโพรซีเดอร์นั้น ๆ เท่านั้น (L0-L7)



รูปที่ 3 ลักษณะของรีจิสเตอร์วินโดว์ที่มีการทับซ้อนกับบางช่วง

ins เป็นกลุ่มรีจิสเตอร์ที่ใช้ในการรับค่าที่ถูกผ่านมาให้กับโพรซีเดอร์ในปัจจุบัน I6 ใช้เก็บค่า FP (Frame Pointer) และถูกใช้ในการชี้ค่าตัวแปรที่อยู่ในเฟรมของสแต็ก (stack frame) FP จะชี้ที่ตำแหน่งของหน่วยความจำที่เดียวกันเสมอตลอดระยะเวลาที่โพรซีเดอร์มีการเอ็กซีคิวต์ ซึ่งไม่เหมือนกับค่าของ SP ที่อาจจะเปลี่ยนขณะที่โพรซีเดอร์มีการเอ็กซีคิวต์ I7 ใช้ในการเก็บตำแหน่งที่จะกลับมาเพื่อโพรซีเดอร์เสร็จจากการเอ็กซีคิวต์

นอกจากนี้รีจิสเตอร์วินโดว์ยังใช้เก็บสถานะของโปรเซสเซอร์ เมื่อมีอินเตอร์รัพต์หรือ trap เกิดขึ้น เมื่อมีการเรียกข้ามโพรซีเดอร์จะทำให้ค่าของ CWP ลดลง เพื่อชี้ไปยังตำแหน่งของวินโดว์ใหม่ให้กับโพรซีเดอร์ที่ถูกเรียกใช้งาน (โพรซีเดอร์ปัจจุบัน) ในกรณีที่มีการเรียกโพรซีเดอร์ติดต่อกัน จนกระทั่งไม่มีวินโดว์ให้ใช้งาน จะต้องมีวิธีการย้ายค่าในวินโดว์ให้มาอยู่ในหน่วยความจำแทน ซึ่งมีขั้นตอนดังนี้ WIM (Window Invalid Mask) คือ รีจิสเตอร์ที่ใช้ในการตรวจสอบจำนวนของวินโดว์ที่สามารถใช้งานได้ โดยปกติภายใน WIM จะมีค่าเป็น "0" ทุกบิต ยกเว้นบิตที่แสดงถึงวินโดว์ที่มีอายุการใช้งานนานที่สุดในปัจจุบันจะมีค่าเป็น "1" ในกรณีที่ไม่มีวินโดว์เหลือให้

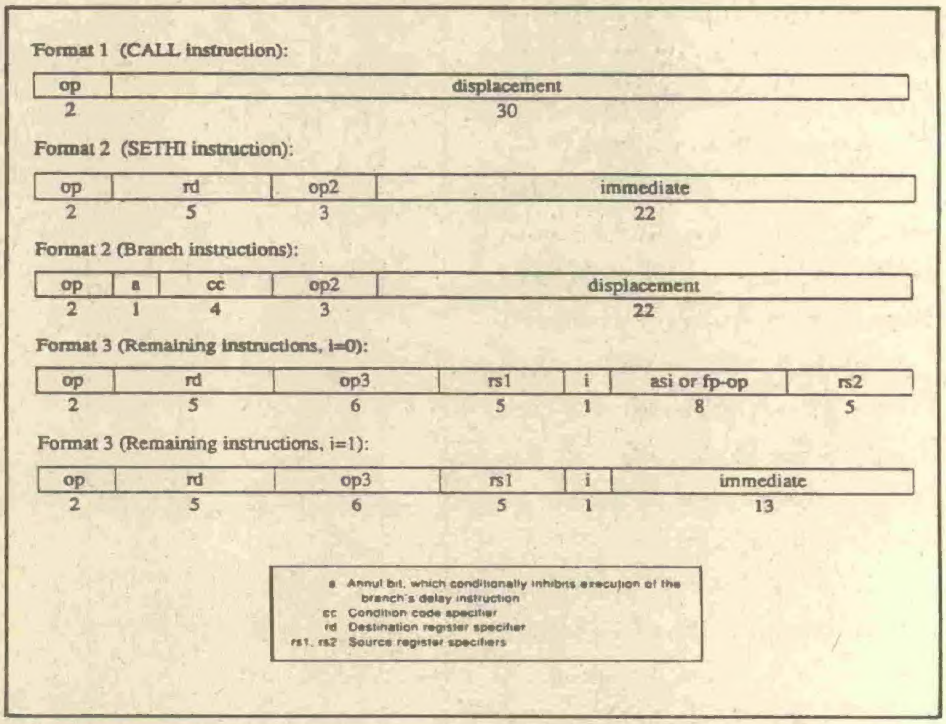
ใช้เมื่อผู้ร้องขอมาจะเกิด trap และโปรเซสเซอร์จะต้องมาจัดการโดยทำการย้ายค่าในวินโดว์ที่มีบิตของ WIM ที่เป็น "1" ชี้อยู่นำไปเก็บไว้ในสแต็กและทำการหมุน "1" ให้ไปอยู่ในตำแหน่งของวินโดว์ที่มีอายุการใช้งานรองลงมา

2.3 ชุดคำสั่งของ SPARC

ชุดคำสั่งของ SPARC มี 3 รูปแบบ โดยใช้เขต op (Op field) เป็นตัวกำหนด ดังแสดงในรูปที่ 4 และมีคำสั่งทั้งหมด 74 คำสั่งดังแสดงในรูปที่ 5 ซึ่งรวมถึงคำสั่งที่เกี่ยวกับ floating-point ด้วย ทั้งนี้ก็เพราะว่าในกรณีที่ไม่มีฮาร์ดแวร์ช่วยในการประมวลผล floating-point จากนั้นจะสามารถจำลองให้ทำได้โดยใช้ซอฟต์แวร์ ชุดคำสั่งทั้ง 3 รูปแบบสามารถอธิบายได้ดังนี้

รูปแบบที่ 1 (format 1) เป็นคำสั่งที่ใช้ในการเรียกโปรแกรมย่อย รูปแบบนี้มี displaement ขนาด 30 บิต ซึ่งสามารถที่จะเรียกไปยังตำแหน่งใด ๆ ก็ได้ภายใน 4 จิกะไบต์ โดยยังคงไว้ซึ่งการทำ word-alignment

รูปแบบที่ 2 (format 2) ประกอบด้วย 1 คำสั่งและ 1 ชุดคำสั่ง คำสั่ง SET เป็นคำสั่งที่ช่วยทำให้การอ้างแอดเดรสใด ๆ ของหน่วยความจำโดยตรงมีความง่ายขึ้น (ที่เป็นเช่นนี้ก็เพราะว่าตามสถาปัตยกรรม SPARC ไม่สามารถอ้างแอดเดรสใด ๆ ของหน่วยความจำโดยตรงได้ภายใน 1 คำสั่ง) โดยคำสั่งนี้จะนำ 22 บิตที่เป็นเขต immediate เข้าไปไว้ที่ตำแหน่งบิตสูงของรีจิสเตอร์ที่กำหนด และทำให้ตำแหน่งบิตต่ำอีก 10 บิต มีค่าเป็น "0"



รูปที่ 4 รูปแบบคำสั่งของ SPARC

หลังจากนั้นคำสั่ง LOAD/STORE สามารถใช้การอ้างแอดเดรสแบบ Index (format 3 เมื่อ i มีค่าเป็น "1") เพื่อกำหนดค่าของ 10 บิตล่างในรีจิสเตอร์มีคำถามที่ต้องการ การกระทำเช่นนี้สามารถที่จะเข้าถึงตำแหน่งใด ๆ ของหน่วยความจำโดยตรงได้โดยใช้ 2 คำสั่ง ชุดคำสั่ง Branch จะมีเขต CC ขนาด 4 บิต เป็นเขตที่บอกชนิดของ branch เช่น BEQ, BLE เป็นต้น displacement ขนาด 22 บิต ทำให้คำสั่ง branch สามารถอ้างไปข้างหน้าหรือย้อนไปข้างหลัง (เมื่อเทียบกับตำแหน่งที่อยู่ในปัจจุบัน) ของหน่วยความจำได้ถึง 8 เมกะไบต์

รูปแบบที่ 3 (format 3) ใช้กับชุดคำสั่งที่นอกเหนือจาก 2 รูปแบบที่กล่าวมาแล้ว เช่น ชุดคำสั่งที่เกี่ยวกับการกระทำทางด้านคณิตศาสตร์และตรรกศาสตร์ เป็นต้น

ชนิดของข้อมูลที่เป็นเลขจำนวนเต็มสำหรับสถาปัตยกรรม SPARC เป็นได้ทั้ง Signed และ unsigned byte, word, halfword, word และ doubleword สำหรับ floating-point จะมี 3 ขนาดคือ 32 บิต (single precision), 64 บิต (double precision) และ 128 บิต (quadruple precision) ซึ่งเป็นไปตามมาตรฐาน IEEE754 สำหรับ quadruple นั้นถูกออกแบบให้ใช้กับสถาปัตยกรรม SPARC version 8

3. บทสรุป

เพื่อให้สถาปัตยกรรม SPARC มีมาตรฐานเดียวกัน หลาย ๆ องค์กรที่มีส่วนร่วมเกี่ยวกับสถาปัตยกรรม SPARC จึงให้ติดตั้งองค์กรที่ไม่หวังผลกำไร (nonprofit organization) ขึ้น เพื่อให้เป็นผู้ดูแลเกี่ยวกับมาตรฐาน

SPARC INSTRUCTION SET

Although true to the minimalist RISC philosophy, the SPARC instruction set exposes certain special features, such as register windowing, to program contral (N/A = not applicable).

oads	Stores	Object to be loaded/storded
DSB/LDSBA	STSB/STSBA	Signed byte
DSH/LDSHA	STSH/STSHA	Signed haltword
DUB/LDUBA	STUB/STUBA	Unsigned byte
DUH/LDUHA	STUH/STUHA	Unsigned haltword
D/LDA	ST/STA	Word
DD/LDDA	STD/STDA	Doubleword
DF	STF	Floating point
DDF	STDF	Double floaling point
DF SR	STFSR	Floating point state register
/A	STDFQ	Double (from Boating point delerted trap queue)
DC	STC	Coprocessor
DDC	STDC	Double coprocessor
DCSR	STCSR	Coprocessor state register
/A	STDCO	Double (from coprocessor delerted trap queue)

at end at instruction - alternate address space

oamic load/store and synchronization

DSTUB/LDSTUBA	Atomic load/store unsigned byte
WAP/SWAPA	Swap register with memory
TBAR	Store batter
LUSH	Flush instruction memory

pecial register manipulation

DASR/WRASR	Read/write ancllary state register
DY/WRY	Read/write register (used in integer multipy/divice)
DPSR/WRPSR	Read/write processor state register
DWIM/WRIWIM	Read/write window unvalid mask
DTBR/WRITER	Read/write trap base rigister

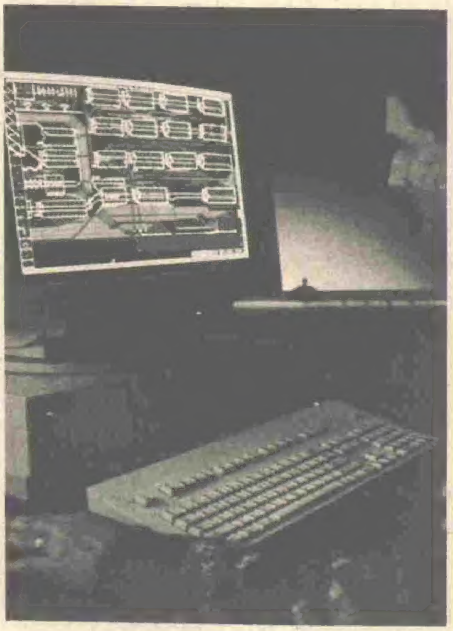
Logical/arithmetic		Branches,calls,Jumps	
AND/ANDcc	AND	Bcc	Branch on integer condition code
ANDN/ANDNcc	AND NOT	FBcc	Branch on floating point condition code
OR/ORcc	OR	CBcc	Branch on coprocessor cc
ORN/ORNcc	OR NOT	CALL	Call and link (return address < 07)
XOR/XORcc	XOR	JMPL	Jump and link (return address < 07)
XNOR/XNORcc	XOR NOT	RETT	Return from trap
ADD/ADDcc	ADD	Ticc	Trap on integer condition code
ADDX/ADDcc	ADD with carry		
SUB/SUBcc	SUB	Register window management	
SUBX/SUBXcc	SUB with borrow		
MULScc	Multiply step	SAVE	Save calier's window
UMUL/UMULcc	Unsinged multiply	RESTORE	Restore calier's window
SMUL/SMULcc	Signed multiply		
UDIV/UDIVcc	Unsigned divice	Miscallaneous	
	SETHI	Set high bits at register	
TADDcc/TADDccTV	Tagged ADD	NOP	No operation
TSUBcc/TSUBccTV	Tagged SUB	UNIMP	Unmpemented (causes trap)
cc = Set condition code			
	based on result	Coprocessor	
ccTV = modify condition codes		CPop	(implementation dependent)
	and trap on overflow		

Floating point	
F(i,s,d,q)TO(i,s,d,q)	Converstons between integer, single, double, quad
FMOvs.FNEGs.FABSs	Move between floating-point register with absolute values or negatives
FSQRT(s,d,q)	Squaroroot
FADD(s,d,q)	Add
FSUB(s,d,q)	Subtract
FMUL(s,d,q)	Multiply
FDIV(s,d,q)	Divide
FCMP(s,d,q)	Compare
FCMPE(s,d,q)	Compare with exception unordered

รูปที่ 5 ชุดคำสั่งของ SPARC

องค์กรนี้มีชื่อว่า "SPARC International (SI)" ถูกก่อตั้งในปี ค.ศ. 1989 โดยเป็นเจ้าของและอนุญาตให้ใช้เครื่องหมายการค้า SPARC ดังแสดงในรูปที่ 6 ซึ่งจะทำการทดสอบทั้งฮาร์ดแวร์, ซอฟต์แวร์ และระบบให้เป็นไปตามมาตรฐาน

โปรเซสเซอร์ SPARC ที่ได้ถูกเปิดตัวในปี ค.ศ. 1987 เป็น scalar processor (โปรเซสเซอร์ที่สามารถเอ็กซีคิวต์ได้เพียง 1 คำสั่งในเวลาหนึ่ง ๆ) แต่ก็มีการนำเทคนิค pipeline เข้ามาช่วยเพื่อทำให้เกิดการทำงานหลาย ๆ อย่างพร้อม ๆ กัน ซึ่งก็คือขบวนการ fetch, devide, execute, memory access และ write back ทำให้ได้อัตราปริมาณงานที่สูงขึ้นกว่าการไม่ใช้เทคนิค pipeline และมีจุดมุ่งหมายที่จะทำให้ CPI (clock per instruction) มีค่าเป็น 1 หรือเข้าใกล้ 1 มากที่สุด



จากการศึกษาพบว่าสมรรถนะของโปรเซสเซอร์ขึ้นอยู่กับคุณลักษณะ 3 ประการ คือ clock cycle CPI และ Instruciton count ทั้ง 3 คุณลักษณะมีความสัมพันธ์ซึ่งกันและกัน ทำให้ไม่สามารถเปลี่ยนแปลงอย่างใดอย่างหนึ่ง โดยไม่มีผลกระทบกับที่เหลือไม่ได้ แต่อย่างไรก็ตาม ในปัจจุบันจะพบว่ามี 3 แนวทางหลัก ๆ ที่พยายามจะพัฒนาให้โปรเซสเซอร์มีสมรรถนะที่ดีขึ้น ซึ่งทั้ง 3 แนวทางนี้ก็นำแต่ละคุณลักษณะข้างต้นมาพัฒนาดังนี้

- การพัฒนาเพื่อให้ใช้ clock cycle น้อยลง (เพื่อ clock rate) ลักษณะเช่นนี้คือเทคนิค superpipelined คือ

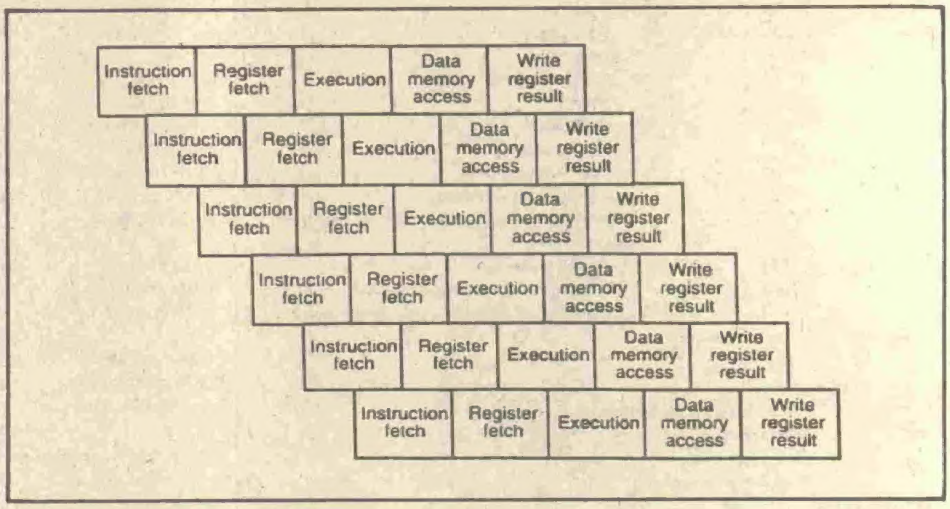
รูปที่ 6 เครื่องหมายการค้าของ SPARC โดยรูปบนเป็น SI member และรูปล่างแสดงถึงมาตรฐานตามข้อกำหนดของ SPARC

การเพิ่มจำนวน pipeline ในโปรเซสเซอร์ให้มากขึ้น ดังแสดงในรูปที่ 7 และใช้กลไกควบคุมการทำงานเพื่อทำให้ทุก ๆ stage ทำงานไปพร้อมกัน

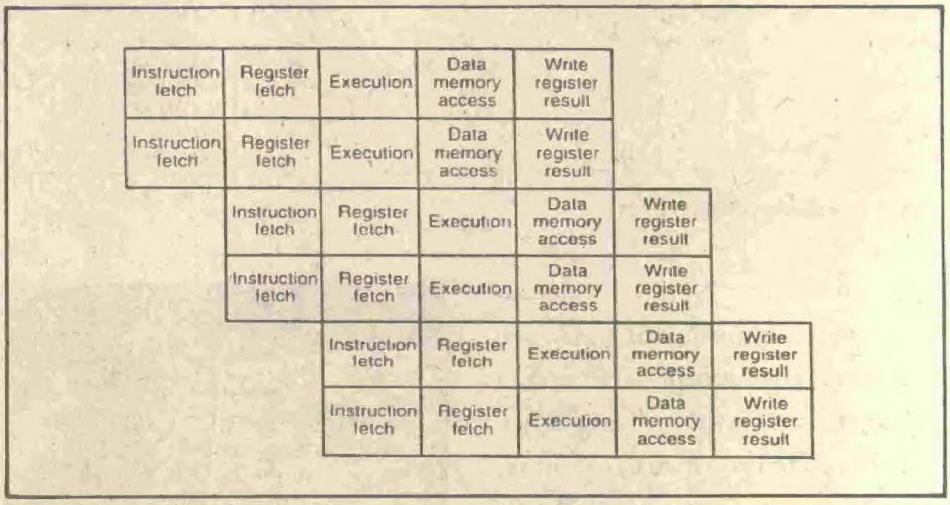
- การพัฒนาเพื่อให้มี CPI ลดลงมา ลักษณะเช่นนี้ก็คือ เทคนิค superscalar คือ การทำให้โปรเซสเซอร์เอ็กซีคิวต์มากกว่า 1 คำสั่งในเวลาเดียวกัน ดังแสดงในรูปที่ 8

- การพัฒนาเพื่อให้มี instruction count ลดลง ลักษณะเช่นนี้คือ เทคนิค VLIW (Very Long Instruction Word) คือการทำให้หลาย ๆ คำสั่งสามารถทำงานขนานกันได้ ซึ่งจะเป็นผลทำให้คำสั่งโดยรวมลดลง

การพัฒนาทั้ง 3 ลักษณะ พบกับปัญหาในลักษณะเดียวกัน ซึ่งก็คือ จะมีวิธีการในการจัดตารางของชุดคำสั่งอย่างไร เพื่อทำให้เกิดการทำงานไปพร้อม ๆ กันหรือขนานกันได้อย่างมีประสิทธิภาพสูงสุด ซึ่งก็คงต้องรอดูกันต่อไปว่าจะเทคโนโลยีที่เกี่ยวกับซอฟต์แวร์หรือฮาร์ดแวร์ใดที่สามารถทำลายสิ่งกีดขวางอันนี้ได้



รูปที่ 7 ลักษณะของ Superpipelined



รูปที่ 8 ลักษณะของ Superscalar

เอกสารอ้างอิง

1. Lazou C., Supercomputers and their Use, Oxford Science Publications, New York, 1988

2. Patterson D.A and Sequen C.H., "A VLSI RISC," IEEE Computer Magazine, Vol. 15, No.9, 1982, pp 8-21

3. Wallich P, "Toward sumpler, faster computer," IEEE Spectrum, Vol 22, No.8, 1985, pp. 35-45

4. Kitagawa K.M. and Leong R.M., "Cypress SPARC-An Advanced Implementation of An Open RISC Architecture," Sun Technology, Vol 2, No.3, 1989, pp. 19-24

5. Popescu V. et al., "The Metaflow Architecture," IEEE Micro Magazine, Vol 11, No.3, 1991 pp. 20-13 and 63-73

6. Hennessy J.H., Patterson D.A, Computer Architecture A Quantitive Approach, Morgan Kanfmann Published Inc., California, 1990

7. Stone H.S. and Cocke J., "Computer Architecture in the 1990," IEEE Computer Magazine, Vol 24, No.9, 1991, pp 30-38

8. Johnson M., Superscalar Microprocessor Design, Prestise-Wall Inc., New Jersey, 1991